The Past, evolving Present and Future of Discrete Logarithm

Antoine Joux, Andrew Odlyzko and Cécile Pierrot

Abstract The first practical public key cryptosystem ever published, the Diffie-Hellman key exchange algorithm, relies for its security on the assumption that discrete logarithms are hard to compute. This intractability hypothesis is also the foundation for the security of a large variety of other public key systems and protocols.

Since the introduction of the Diffie-Hellman key exchange more than three decades ago, there have been substantial algorithmic advances in the computation of discrete logarithms. However, in general the discrete logarithm problem is still considered to be hard. In particular, this is the case for the multiplicative group of finite fields with medium to large characteristic and for the additive group of a general elliptic curve.

This paper presents a current survey of the state of the art concerning discrete logarithms and their computation.

1 Introduction

1.1 The Discrete Logarithm Problem

Many popular public key cryptosystems are based on discrete exponentiation. If G is a multiplicative group, such as the group of invertible elements in a finite field or the group of points on an elliptic curve, and g is an element of G, then g^x is the discrete exponentiation of base g to the power x. This operation shares basic properties with ordinary exponentiation, for example $g^{x+y} = g^x \cdot g^y$. The inverse operation is, given h in G, to determine a value of x, if it exists, such that $h = g^x$. Such a number x is called a discrete logarithm of h to the base g, since it shares many properties with the ordinary logarithm. If, in addition, we require some normalization of x to limit the possible answers to single valid value we can then speak of *the* discrete logarithm of h.

Antoine Joux
CryptoExperts, Paris, France
Chaire de Cryptologie de la Fondation de l'UPMC, Paris, France
Sorbonne Universités, LIP6, UMR 7606, UPMC Univ Paris 06, France
e-mail: `Antoine.Joux@m4x.org`

Andrew Odlyzko
School of Mathematics, University of Minnesota, Minneapolis, MN 55455, USA
e-mail: `odlyzko@umn.edu`

Cécile Pierrot
DGA/CNRS, Sorbonne Universités, LIP6, UMR 7606, UPMC Univ Paris 06, France
e-mail: `Cecile.Pierrot@lip6.fr`

Indeed, without such a normalization, x is not unique and is only determined modulo the order of the element g.

Assume for simplicity that G is a cyclic group generated by g, and that the notation $\log_g(h)$ denotes a value such that $h = g^{\log_g(h)}$. Then, as with ordinary logarithms, there is a link between multiplication of elements and addition of logarithms. More precisely, we have:

$$\log_g(h \cdot j) \equiv \log_g(h) + \log_g(j) \mod |G|.$$

We say that we solve the discrete logarithm problem (DLP) in G if, given any element g^x in G we are able to recover x. To normalize the result, we usually ask for x to be taken in the range $0 \leqslant x < |G|$. In many applications, in particular in cryptography, it is sufficient to be able to solve this problem in a substantial fraction of cases. (The usual theoretical standard is that this fraction should be at least the inverse of a polynomial in the logarithm of the size of the group.)

The main interest of discrete logarithm for cryptography is that, in general, this problem is considered to be hard. The aim of this paper is to provide state-of-the-art information about the DLP in groups that are used for cryptographic purposes. It gives pointers to the latest results and present observations about the current status and likely future of the DLP.

1.2 Applications of Discrete Logarithms

In some sense, the discrete logarithm has a long history in number theory. It is just an explicit way to state that an arbitrary cyclic group containing N elements is isomorphic to $(\mathbb{Z}_N, +)$. Still, before the invention of the Diffie-Hellman protocol the problem of efficiently computing discrete logarithms attracted little attention. Perhaps the most common application was in the form of Zech's logarithm, as a way to precompute tables allowing faster execution of arithmetic in small finite fields.

The role of the DLP in cryptography predates Diffie-Hellman. Indeed, the security of secret-key cryptosystem involving linear feedback-shift registers (LFSR) is closely related to the computation of discrete logarithms in finite fields of characteristic two. More precisely, locating the position where a given subsequence appears in the output of an LFSR is, in fact, a discrete logarithm problem in the finite field defined by the feedback polynomial[1].

The main impetus to intensive study of discrete logarithms came from the invention of the Diffie-Hellman method in 1976 [DH76]. Much later, the introduction of pairing in cryptography in 2000 (journal versions [Jou04, BF03]) increased the level of attention on some atypical finite fields, with composite extension degrees and/or medium-sized characteristic.

Diffie-Hellman Key Exchange

Let us recall the first practical public key technique to be published, which is still widely used, the Diffie-Hellman key exchange algorithm. The basic approach is as follows. If Alice and Bob wish to create a common secret key, they first agree on a cyclic group G and a generator g of this group.[2] Then, Alice chooses a random integer a, computes g^a and sends it to Bob over a public channel, while Bob chooses a random integer b and sends g^b to Alice. Now Alice and Bob can both compute a common value, which then serves as their shared secret:

$$(g^b)^a = g^{a \cdot b} = (g^a)^b$$

The security of this system depends on the assumption that an eavesdropper who overhears the exchange, and thus knows g, g^a, and g^b, will not be able to compute the shared secret. In particular, this hypothesis assumes that the eavesdropper is unable to solve the discrete logarithm

[1] Assuming that it is irreducible, which is usually the case.

[2] The group G and generator g can be the same for many users, and can be part of a public standard. However, that can lead to a reduction in security of the system.

problem in G. Indeed, if the DLP for this group is solvable, he can compute either a or b and recover the shared secret $g^{a \cdot b}$. However, it is not known whether the problem of computing g^{ab} given g, g^a, and g^b, which is known as the computational Diffie-Hellman problem (CDH) is equivalent to the computation of discrete logarithms. Moreover, to prove the security of many cryptographic protocols, it is often necessary to consider the associated decision problem: given g, g^a, g^b and h, decide whether h is the correct value of g^{ab} or not. This latest problem is called the decision Diffie-Hellman problem (DDH).

There are also many generalized computational and decision problems somehow related to the discrete logarithm problem that have been introduced as possible foundations for various cryptosystems. Since it is not easy to compare all these assumptions, in an attempt to simplify the situation, Boneh, Boyen and Goh [BBG05] have proposed the *Uber assumption* which subsumes all these variations and can be proven secure in the generic group model (see Section 2.5).

However, the discrete logarithm problem itself remains fundamental. Indeed from a mathematical viewpoint, it is a much more natural question than the other related problems and, in practice, none of these other problems has ever been broken independently of the DLP. Since the introduction of the Diffie-Hellman key exchange, this concern has motivated a constant flow of research on the computation of discrete logarithms.

Another extremely important assumption in the above description is that the eavesdropper is passive and only listens to the traffic between Alice and Bob. If the attacker becomes active, then the security may be totally lost, for example if he can mount a *man-in-the-middle* attack where he impersonates Bob when speaking to Alice and conversely. This allows him to listen to the decrypted traffic. To avoid detection, the attacker forwards all messages to their intended recipient after reencrypting with the key that this recipient has shared with him during the initial phase. One essential issue when devising cryptosystems based on discrete logarithms is to include safety measures preventing such active attacks.

Other Protocols

After the invention of the RSA cryptosystems, it was discovered by ElGamal [Gam85] that the discrete logarithm problem can be used not only for the Diffie–Hellman key exchange, but also for encryption and signature. Later Schnorr [Sch89] gave an identification protocol based on a zero-knowledge proof of knowledge of a discrete logarithm, which can be turned into Schnorr's signature scheme using the Fiat-Shamir transform [FS86].

There are many more cryptosystems based on the discrete logarithm problem which will not be covered here. However, let us mention Paillier's encryption [Pai99]. This system works in the group $\mathbb{Z}_{N^2}^*$, where $N = pq$ is an RSA number of unknown factorization. In particular, this is an example of a discrete logarithm based cryptosystem that works within a group of unknown order. This system possesses an interesting property, in that it is additively homorphic; the product of the Paillier encryption of two messages is an encryption of their sum.

Another very interesting feature of discrete logarithms is the ability to construct key exchange protocols with additional properties, such as authenticated key exchange, which embed the verification of the other party identity within the key exchange protocol. Perfect forward secrecy, in which disclosure of long-term secrets does not allow for decryption of earlier exchanges, is also easy to provide with schemes based on discrete logarithms. For example, in the Diffie-Hellman key exchange, Alice's secret a and Bob's secret b are ephemeral, and so is the shared secret they are used to create, and (if proper key management is used) are discarded after the interaction is completed. Thus an intruder who manages to penetrate either Alice's or Bob's computer would still be unable to obtain those keys and decrypt their earlier communications. It is also possible to mix long-term secrets, i.e. private keys, and ephemeral secrets, in order to simultaneously provide perfect forward secrecy and identity verification.

A Powerful Extension: Pairing–Based Cryptography

Besides the Diffie-Hellman key exchange, a natural question to ask is whether there exists a three-party one-round key agreement protocol that is secure against eavesdroppers. This question remained open until 2000 when Joux [Jou04] devised a simple protocol that settles this question using bilinear pairings. Until then, building a common key between more than two users required two rounds of interaction. A typical solution for an arbitrary number of users is the Burmester-Desmedt protocol [BD94].

The one-round protocol based on pairing works as follows. If Alice, Bob and Charlie wish to create a common secret key, they first agree on $G_1 = \langle P \rangle$ an additive group with identity $\mathcal{O}$, a multiplicative group G_2 of the same order, with identity 1 and a bilinear pairing from G_1 to G_2. Let us recall the definition :

Definition 1.1. A symmetric bilinear pairing[3] on (G_1, G_2) is a map

$$e : G_1 \times G_1 \to G_2$$

satisfying the following conditions:

1. e is bilinear: $\forall\, R, S, T \in G_1,\ e(R+S, T) = e(R,T) \cdot e(S,T)$
   and $e(R, S+T) = e(R,S) \cdot e(R,T)$.
2. e is non-degenerate: If $\forall\, R \in G_1,\ e(R,S) = 1,$ then $S = \mathcal{O}$.

Alice randomly selects a secret integer a modulo the order of G_1 and broadcasts the value aP to the other parties. Similarly and simultaneously, Bob and Charlie select their one secret integer b and c and broadcast bP and cP. Alice (and Bob and Charlie respectively) can now compute the shared secret key :

$$K = e(bP, cP)^a = e(P,P)^{abc}$$

We know that the security of DH-based protocols often relies on the hardness of the CDH and DDH problems. Likewise, the security of pairing-based protocols depends on the problem of computing $e(P,P)^{abc}$ given P, aP, bP and cP, which is known as the computational bilinear Diffie-Hellman problem (CBDH or simply BDH). This problem also exists in its decisional form (DBDH). However, little is known about the exact intractability of the BDH and the problem is generally assumed to be as hard as the DLP in the easier of the groups G_1 and G_2. Indeed, if the DLP in G_1 can be efficiently solved, then an eavesdropper who wishes to compute K can recover a from aP and then compute $e(bP, cP)^a$. Similarly, if the DLP in G_2 can be efficiently solved, he could recover bc from $e(bP, cP) = e(P,P)^{bc}$, then compute bcP and finally obtain K as $e(aP, bcP)$.

One consequence of the bilinearity property is that the DLP in G_1 can be efficiently reduced to the DLP in G_2. More precisely, assume that Q is an element of G_1 such that $Q = xP$, then we see that $e(P,Q) = e(P, xP) = e(P,P)^x$. Thus, computing the logarithm of $e(P,Q)$ in G_2 (to the base $e(P,P)$) yields x. This reduction was first described by Menezes, Okamoto, and Vanstone [MOV93] to show that supersingular elliptic curves are much weaker than random elliptic curves, since the discrete logarithm problem can be transfered from a supersingular curve to a relatively small finite field using pairings.

After the publication of the Menezes, Okamoto, and Vanstone result, cryptographers started investigating further applications of pairings. The next two important applications were the identity-based encryption scheme of Boneh and Franklin [BF03] and the short signature scheme of Boneh, Lynn and Shacham [BLS04]. Since then, there has been a tremendous activity in the design, implementation and analysis of cryptographic protocols using bilinear pairings on elliptic curves, and also on more general abelian varieties, for example, on hyperelliptic curves.

[3] In general, asymmetric pairings are also considered. For simplicity of presentation, we only describe the symmetric case.

1.3 Advantages of Discrete Logarithms

A large fraction of the protocols that public key cryptography provides, such as digital signatures and key exchange can be accomplished with RSA and its variants. Pairing-based cryptosystems are a notable exception to this general rule. However, even for classical protocols, using discrete logarithms instead of RSA as the underlying primitive offers some notable benefits.

Technical Advantages

Smaller key sizes

The main advantage of discrete logarithms comes from the fact that the complexity of solving the elliptic curve discrete logarithm problem (ECDLP) on a general elliptic curve is, as far as we know, much higher than factoring an integer of comparable size. As a direct consequence, elliptic curve cryptosystems currently offer the option of using much smaller key sizes than would be required by RSA or discrete logarithms on finite fields to obtain a comparable security level.

In truth, the key size reduction is so important that it more than offsets the additional complexity level of elliptic curve arithmetic. Thus, for the same overall security level, elliptic curve systems currently outperform more classical systems.

Perfect forward secrecy

When using RSA to setup a key exchange, the usual approach is for one side to generate a random secret key and send it to the other encrypted with his RSA public key. This grants, to an adversary that records all the traffic, the ability to decrypt every past communications, if he ever gets hold of the corresponding private key.

By contrast, as we have already mentioned in the introduction, a correctly designed key exchange protocol based on the discrete logarithm problem can avoid this pitfall and achieve perfect forward secrecy, thus preventing an adversary to decrypt past communications [DOW92].

Algorithmic diversity

Cryptographers have learned from history that it is unwise to base security on a single assumption, as its violation can lead to simultaneous breakdown of all systems. For this reason it is important to have a diversity of cryptosystems and have candidate replacement systems. Schemes based on discrete logarithms provide an alternative to those derived from RSA and other algorithms whose security depends on difficulty of integer factorization.

However, we should note that both integer factorizations and discrete logarithms would be easy to obtain from quantum computers. Hence it is important to investigate even more exotic cryptosystems, such as those based on error-correcting codes and lattices.

The paper is organized as follows. Section 2 deals with generic algorithms, *i.e.* those that assume no special knowledge about the underlying group and consider group operations as black boxes. By contrast, Section 3 presents the Index Calculus Method, a very useful framework to obtain a family of algorithms that make extensive use of specific knowledge of the group. Section 4 presents concrete algorithms to solve the DLP in finite fields and Section 5 describes the state-of-the-art about the DLP on algebraic curves.

2 Generic Results

This section discusses some general results for discrete logarithm that assume little knowledge of the group. In the most general case, we only ask for a group whose elements can be represented in a compact way and whose law is explicitly given by an efficient algorithm. We also consider the case where the order of the group and possibly its factorization are also given. This case is interesting because for many groups that are considered in practice, this information is easily obtained. Typically, for an elliptic curve, the group order is efficiently found using point counting algorithms. Moreover, system designers usually choose curves whose order is a small multiple of a prime, in which case, factoring the order becomes easy.

Throughout the section, we use the same notations as in Section 1.1. First, we describe some general complexity results that relates the hardness of the discrete logarithm problem to classical complexity theoretical classes. Second, assuming that the factorization of the order of G is given, we show that computing discrete logarithms in G is no harder than computing discrete logarithms in all subgroups of G of prime order. This is the outcome of the Pohlig–Hellman algorithm [PH78], which is a constructive method to compute discrete logarithms in the whole group from a small number of computations of discrete logarithms in the subgroups of prime order. We also describe Pollard's Rho algorithm [Pol78] that allows the computation of discrete logarithms in a group G in $O(\sqrt{|G|})$ operations. Combining Pohlig–Hellman with Pollard's Rho essentially[4] permits the computation of discrete logarithms in time $O(\sqrt{p})$, where p is the largest prime factor of the order of a generic group.

Finally, we discuss the issue of computing many independent discrete logarithms in the same generic group, amortizing part of the computation cost; we also briefly present the generic group model as proposed by Shoup in [Sho97b] and the lower bound on the complexity that is related to it.

2.1 Complexity classes

In order to describe the exact level of hardness of a computational problem, the main approach is to describe the complexity classes the problem belongs to. To this end, the traditional approach is to work with decision problems, *i.e.*, problems with a yes/no answer. Since the discrete logarithm problem itself is not a decision problem, the first step is to introduce a related decision problem whose hardness is essentially equivalent to computing discrete logarithms. This can be done in many ways, for example, let us consider the following problem.

Problem 2.1 (Log Range Decision). **Given a cyclic group G and a triple (g, h, B):**

- Output **YES** if there exists $x \in [0 \cdots B]$ such that $h = g^x$.
- Otherwise output **NO**.

An algorithm or oracle that solves this problem can be used to compute discrete logarithms using a binary search. This requires a logarithmic number[5] of calls to *Log Range Decision.* As a consequence, the hardness of *Log Range Decision* is essentially the same as the hardness of the Discrete Logarithm Problem itself.

Log Range Decision is in NP ∩ co-NP

To show that the problem is in NP, assume that there exists $x \in [0 \cdots B]$ such that $h = g^x$, then x itself is a witness to this fact which is easily tested in polynomial time.

When g is a generator of G and $|G|$ is known, giving a possible discrete logarithm of h to the base g is also a satisfying witness to prove that the answer is NO. Thus, in this simple case, the problem

[4] If $|G|$ is a product of many small primes, possibly with multiplicity, this claim does not hold. However, this is not an interesting case for cryptographic purposes.

[5] In other words, the number of oracle calls is a polynomial in the bitsize of the answer.

belongs to co-NP. However, in general, the situation is more complex: we need a generator[6] g_0 of G together with $|G|$ and its factorization to prove this fact. With g_0 in hand, the discrete logarithms of both g and h to the base g_0 suffice to determine whether h belongs to the subgroup generated by g and, if needed, to prove that none of the discrete logarithms of h to the base g belong to $[0 \cdots B]$. As a consequence, even in the general case, Log Range Decision is in co-NP.

Log Range Decision is in BQP

Another very important complexity theoretic result about the computation of discrete logarithms is that there exists an efficient *quantum* algorithm invented by Shor [Sho97a]. This algorithm works for arbitrary groups, assuming that the group operation can be computed efficiently. It is based on the Quantum Fourier transform and belongs to the complexity class **BQP** (Bounded-error Quantum Polynomial time) that corresponds to polynomial time computation on a quantum computer with a bounded error probability[7].

Computing $|G|$ using a discrete logarithm computation

When considering the discrete logarithm problem, we often assume that the group order $|G|$ is known. One justification is that it is often the case which the groups that are used in cryptography. Here, we point out another reason. When the discrete logarithm problem becomes easy in a group, it is possible to compute $|G|$ using a discrete logarithm computation. Assume that we are only given the bitsize of $|G|$, i.e. that we know that $|G| \in [2^{n-1}, 2^n - 1[$. In this context, given g a generator of G, we see that:

$$|G| = 2^n - \log_g(g^{2^n}).$$

Average case hardness and random self-reducibility

When using hard problems to build cryptosystems, one important issue is to be sure that randomly generated instances of the problem are practically hard. This deviates from the standard definition of hardness in complexity. In cryptography, a problem that admits hard instances is not enough, we need the problem to be hard not only in its worst case, but also in its average case, and also usually in most cases.

Concerning the discrete logarithm problem, we have a very nice property, *random self-reducibility* introduced in [AFK89]. This property shows that any instance of the DLP can be rerandomized into a purely random instance. As a consequence, if the DLP is easy in the average case it is also easy in the worst case. Conversely, if there exists hard instances of the DLP in some group G, then the DLP is hard for random instances in G.

The reduction works as follow, assume that we are given an oracle that solves the DLP in G for random instances and some fixed instance of the problem, $h = g^x$. Choose an integer r modulo $|G|$ uniformly at random and define $z = hg^r$, then z follows a uniform random distribution in G. If the given oracle can compute $\log_g(z)$, we recover x from the relation $x \equiv \log_g(z) - r \pmod{|G|}$.

2.2 Pohlig–Hellman

Let G be a group of order n, g a generator, and h the element for which we want to compute the discrete logarithm x. We suppose further that we know the factorization of n:

[6] Since there are many distinct generators of G, in fact $\varphi(G)$, g_0 is easy to find by testing random candidates.
[7] Typically, an error probability of 1/3 can be used in the formal definition of BQP.

$$n = \prod_{p_i | n} {p_i}^{e_i}.$$

The Pohlig-Hellman algorithm permits us to reduce the DLP in G to DLPs in cyclic groups of prime order p_i. We proceed in two phases:

1. First we reduce the DLP in G to DLPs in groups with orders a power of the primes p_i involved in the factorization of n. For each p_i, we set:

$$n_i = \frac{n}{{p_i}^{e_i}}, \qquad g_i = g^{n_i} \quad \text{and} \quad h_i = h^{n_i}.$$

   So ${g_i}^{{p_i}^{e_i}} = g^n = 1$ and the order of g_i is exactly ${p_i}^{e_i}$. Moreover $g_i^x = g^{n_i x} = h^{n_i} = h_i$. Thus h_i belongs to the subgroup of order $p_i^{e_i}$ generated by g_i. More precisely, h_i can be considered as the projection of the element we want the discrete logarithm on the subgroup generated by g_i. Let us call x_i the discrete logarithm of h_i to the base g_i. We then have

$$x \equiv x_i \mod p_i^{e_i}.$$

   Since the $p_i^{e_i}$ are pairwise coprime, if we know all the x_i, a simple application of the Chinese Remainder Theorem permits us to recover x.
2. A further simple reduction shows that solving the DLP in a group of prime order allows to solve the DLP in groups with orders that are powers of that prime.

To conclude, what has to be kept in mind is that computing discrete logarithms in G is no harder than computing discrete logarithms in all subgroups of prime order in G.

2.3 Discrete Logarithms in G in $O\left(\sqrt{|G|}\right)$.

There are several methods for computing discrete logarithms in a group G in about $\sqrt{|G|}$ operations. The first and best known of these is the Shanks Baby Step/Giant Step technique.

Baby Step/Giant Step

Let n be the order of G, or even an upper bound of $|\langle g \rangle|$, and h be the element for which we want to compute the discrete logarithm x. Let m be equal to $\lceil \sqrt{n} \rceil$. If we let q and r be such that $x = qm + r$ with $0 \leqslant r, q < m$, which is possible thanks to the size of m compared to n, then it is clear that finding x is exactly the same as recovering q and r. First we remark that we have:

$$(g^m)^q = (g^{mq} g^r) g^{-r} = g^{mq+r} g^{-r} = h g^{-r}. \tag{1}$$

We create the first list:

$$\text{Baby } = \{(hg^{-r}, r) | 0 \leqslant r < m\}$$

We call it the Baby list, because we multiply each step by the inverse of g (which is considered to be small). If, by good luck, there exists a couple $(1, r')$ in this set, we have obtained $hg^{-r'} = 1$ and thus $x = r'$. If not, we create another list:

$$\text{Giant } = \{((g^m)^q, q) | 0 \leqslant q < m\}$$

called the Giant list because we multiply in this case each step by g^m. We sort the two lists to find a collision on the two first elements of each pair. When we obtain $(hg^{-r}, r) \in$ Baby such that $(g^m)^q = hg^{-r}$, thanks to (1), we have found q and r and thus x.

Checking for equality in two sorted lists of m entries each can be done in linear time (assuming that the representations of elements are compact enough). Hence the running time of the algorithm is dominated by the arithmetic required to compute the two lists and the time to sort them. This algorithm is deterministic and solves the DLP in $\widetilde{O}\left(\sqrt{n}\right)$ operations[8].

Pollard's Rho Algorithm

This algorithm runs in time comparable to the Shanks method, $O\left(\sqrt{n}\right)$ operations, but has the advantage that it is practically memoryless. Unlike the Shanks algorithm, though, it is probabilistic, not deterministic. It was proposed by John M. Pollard in 1978 [Pol78] and works as follows.

Let us imagine that we have a partition of G into three subsets of roughly equal size A_1, A_2 and A_3 . We define the map f by:

$$f(b) = \begin{cases} gb \text{ if } b \in A_1 \\ b^2 \text{ if } b \in A_2 \\ hb \text{ if } b \in A_3 \end{cases}$$

We take now a random integer x_0 in $\{1, \cdots, n\}$ and we compute $b_0 = g^{x_0}$. We consider the sequence $b_{i+1} = f(b_i)$. The algorithm relies on two facts. First, for each i we can rewrite b_i as

$$b_i = g^{x_i} h^{y_i} \tag{2}$$

where $(x_i)_i$ and $(y_i)_i$ are given by the initial choice of x_0, $y_0 = 0$ and :

$$x_{i+1} = \begin{cases} x_i + 1 \mod n \text{ if } b_i \in A_1 \\ 2x_i \mod n \text{ if } b_i \in A_2 \\ x_i \text{ if } b_i \in A_3 \end{cases} \quad \text{and} \quad y_{i+1} = \begin{cases} y_i \text{ if } b_i \in A_1 \\ 2y_i \mod n \text{ if } b_i \in A_2 \\ y_i + 1 \mod n \text{ if } b_i \in A_3 \end{cases}$$

Second, since we are computing a sequence in a finite group, there exist two integers $i \geqslant 0$ et $k \geqslant 1$ such that we have a collision $b_i = b_{i+k}$ (in practice we search collision of the form $b_i = b_{2i}$). Thanks to equation (2) we have:

$$g^{x_i} h^{y_i} = g^{x_{i+k}} h^{y_{i+k}}$$

which yields a linear equation for $\log_g(h)$:

$$x_i - x_{i+k} \equiv \log_g(h)(y_{i+k} - y_i) \mod n.$$

If we can invert $y_{i+k} - y_i$ modulo n then we can recover the discrete logarithm of h. If $y_{i+k} - y_i$ is not invertible, we need to remember that Pollard Rho is usually used as a subroutine of Pollig–Hellman, which means that n is usually prime. As a consequence, the only option is to restart a different instance of computation, for example using another choice for x_0.

The Pollard Rho algorithm can be implemented so that it requires only $O(1)$ elements in memory and $O\left(\sqrt{n}\right)$ operations. Some practical improvements of this algorithm are presented in [Tes00, BLS11, CHK12].

In practice, computations of discrete logarithms using generic algorithms use a combination of the Pohlig-Hellman and Pollard Rho algorithms. Depending on the computer architecture used for the computations, there exists alternatives to Pollard's Rho that are sometimes more appropriate (see the next Section). However, the overall complexity using these algorithms remains $O\left(\sqrt{p}\right)$ where p is the largest prime dividing the order of the group. In fact, Section 2.5 shows that generic group algorithms cannot outperform this complexity.

2.4 Scalability of Generic Discrete Logarithm Algorithms

From a purely theoretical viewpoint, a $O(\sqrt{|G|})$ algorithm that only uses a constant amount of memory is a very fine solution. However, for practical purposes, it is very useful to know whether

[8] As usual, the $\widetilde{O}$ notation $\widetilde{O}(n)$ is a shorthand for $O(n \log^{\alpha} n)$ for an arbitrary value of α.

such a computation can be distributed on a parallel computer or a network of independent computers. Indeed, this scalability issue often decides whether a computation is feasible or not.

In this setting, it is very useful to replace cycle finding algorithms by algorithms based on the distinguished point technique. According to [Den82, p. 100] the idea of the distinguished point technique was proposed by Rivest. Quisquater and Delescaille [QD89] used the technique to find collisions in the DES algorithm. The in-depth study made by van Oorschot and Wiener [vOW99] shows how the technique can be used in order to efficiently take advantage of parallelism for collision search.

Basically, the main idea of the distinguished point technique is to build chains of computations, starting from a random value and iterating a fixed function f to compute a chain of successors. Denoting the starting point x_0, we iteratively compute $x_{i+1} = f(x_i)$. We abort the computation when encountering a point x_N that satisfies some distinguished point property. Typically, this property is taken to be that the representation of x_N starts with a specified number of '0' bits. We then store the triple (x_0, x_N, N). Recall that as in Pollard Rho, we wish to find a collision of f in order to compute the desired discrete logarithm. With the distinguished point technique, any collision between two distinct chains ensures that the two chains terminate at the same distinguished point. Conversely, given two chains ending at the same distinguished point, recomputing the two chains from their respective starting points, accounting for the length difference, usually leads to an explicit collision. Since the initial computations of chains are independent from each other, it is extremely easy to distribute them over a large number of distinct computers.

Recently, using a slight variation of this distinguished point technique, it was shown in [FJM13] that given L independent discrete logarithms to compute in the same group $|G|$, the computation can be achieved in time $O(\sqrt{L|G|})$ rather than $O(L\sqrt{|G|})$. Similar results were already known under the condition $L \leq O(|G|^{1/4})$ [KS01].

2.5 The Generic Group Model

In 1997, Shoup [Sho97b] introduces a theoretical framework to study the complexity of generic algorithms: the generic group model. In this model, he shows that any generic algorithm must perform $\Omega(\sqrt{p})$ group operations, where p is the largest prime dividing the order of the group. Since this lower bound essentially[9] matches the known upper bound, the generic group model emphasizes the fact that currently known generic algorithms for computing discrete logarithms are optimal.

In a nutshell, in the generic group model, group elements are identified by unique but arbitrary encodings. As a consequence, it is not possible to exploit any special properties of the encodings and group elements can only be operated on using an oracle that provides access to the group operations.

One frequently encountered criticism of the generic group model is that it suffers from the same weaknesses as the random oracle model, which is considered with suspicion by many cryptographers. Namely, in these models, there exists secure protocols that cannot be securely instantiated [Den02, CGH00].

3 Index Calculus Method

The results from the generic group model no longer apply when extra information about the group structure is known. Indeed, this extra information can then be used to obtain faster algorithm. The most important example is the Index Calculus Method which uses this additional knowledge to provide subexponential algorithms.

Though the Index Calculus Method works both for factoring and for discrete logarithm, here we only consider its application to discrete logarithm computations.

[9] Up to logarithmic factors.

3.1 General Description

The basic idea of Index Calculus algorithms relies on three main steps: the sieving phase (also called the relation collection phase), the linear algebra phase and the individual logarithm phase. Basically, the first phase creates relations between the logarithms of elements belonging to a small subset of the considered group, the second one recovers those logarithms and the last one permits to obtain the logarithm of any arbitrary element by relating it to the logarithms obtained during the first two phases. Those three steps works as follows:

1. **Sieving Phase or Relation Collection Phase** For simplicity, assume that G is a cyclic group generated by g. We want to create a large number of multiplicative relations between elements belonging to a subset of the group G. This subset is usually constructed by selecting elements which can be considered to be *small*, in some sense that depends on the context. This subset of G is usually called the smoothness basis or the factor basis. Let $\{g_i,\, i \in I\}$ denote this smoothness basis and consider a relation of the form:
$$\prod_{i\in I} {g_i}^{m_i} = \prod_{i\in I} {g_i}^{n_i}. \tag{3}$$
Then, taking the discrete logarithms of the two sides, we deduce:
$$\sum_{i\in I} m_i \log_g g_i \equiv \sum_{i\in I} n_i \log_g g_i \mod |G|.$$
This becomes a linear equation between the logarithms of the g_i viewed as formal unknowns. We stop the sieving phase once we have collected enough such linear equations to obtain a system of codimension 1.
2. **Linear Algebra Phase** The aim of the linear algebra step is to solve the previous system of linear equations. Thus, we get at the end of this phase all the discrete logarithms of the smoothness basis[10].
A very important observation that naturally applies in most Index Calculus algorithms is that the equations produced during the relation collection phase are very sparse. This is extremely important, because sparse system can be solved using special algorithms which are much faster than general linear system algorithms. This is detailed in Section 3.4.
3. **Individual Logarithm Phase** To really solve the discrete logarithm problem in G, we should be able to compute the logarithm of any arbitrary element z of G. Roughly, the goal of this last phase is to decompose z into products of other elements, which can in some sense be considered smaller than z and iterate until z is finally expressed as a product of elements belonging to the smoothness basis. Plugging the values of the discrete logarithms obtained during the first two phases in this expression yields the logarithm of z.

3.2 Collection of Relations

In order to design Index Calculus algorithms, we thus need to construct multiplicative relations as in (3).

The simplest approach for discrete logarithms modulo a prime p is to take a random integer a, compute $u \equiv g^a \mod p$ for u an integer such that $1 \leqslant u \leqslant p-1$, and check whether
$$u = \prod q_i$$
where the q_i are primes satisfying $q_i < B$ for some bound B. When the above congruence holds, we say that u is B-smooth and we call B the smoothness bound. For most values of a, u will not be smooth, and so will be discarded. However, even with this primitive approach, one can

[10] Or at least, a large fraction of these logarithms. Indeed, depending on the exact properties of the relation collection phase, a few elements of the smoothness basis might possibly be missing.

obtain running time bounds of the form[11] $L_p(1/2, c)$. Moreover, this approach provides a provable although probabilistic algorithm for solving the DLP in many finite fields.

Though this remains an interesting algorithm because it is at once simple and rigorous, it is possible to devise better algorithms with other strategies. One key idea is to represent the group G in which we want to compute discrete logarithms in two different but compatible ways. In other words, we want to be able to draw a commutative diagram like the one presented in Figure 1. With this representation in hand, for all x in E, we can get two elements in G related in an algebraic way. Thanks to commutativity, we have an equality in the group G:

$$\varphi_1(\psi_1(x)) = \varphi_2(\psi_2(x)).$$

However, for this to be useful, we need to have a way to select some special relations among those created. To this end, we choose a small set in each intermediate set E_1 and E_2 of the diagram. Once these are chosen, in the sieving phase we keep only relations that involve elements of these two small sets and no others. We call the smoothness base (or factor base) the subset of G consisting of elements that can be obtained trough these two small subsets[12] of E_1 and E_2. The Number Field Sieve [Sch00, Gor93, JLSV06] and the Function Field Sieve [AH99, JL06] that have complexity of the form $L_{p^n}(1/3, c)$ both follow this general strategy. They are heuristic algorithms in that their analyses depend on plausible assumptions, but ones that have not been proved rigorously. Despite the fact that they share a common algorithmic structure, there is a major difference between the NFS and the FFS. The former algorithm is based on multiplicative relations between algebraic integers in number fields while the latter works in function fields. At the bottom level, this means that one algorithm needs to factor integers while the other factors polynomials. This is a major difference since polynomials are much easier to factor than integers and also have more systematic properties which have been used in the recent algorithms reported in the next paragraph.

A small change in just the sieving phase can lead to a substantial improvement in the complexity of an algorithm. In fact, recent progress in the Index Calculus method for the discrete logarithm problem has come from better collections of relations. However, the notion of sieving tends to disappear since the new algorithms proposed to solve the DLP in finite fields with small characteristic rely on a new trick that directly creates those relations. Those new methods developed in paragraph 4.2 has recently yielded complexities in $L_{p^n}(1/4, c)$ [Jou13b] for finite fields with a small characteristic. With an additional improvement made this time in the individual logarithm phase, this has led to a heuristic quasi-polynomial algorithm [BGJT13], again for large fields of small characteristic.

$$\begin{tikzcd} & E \arrow[dl, "\psi_1"'] \arrow[dr, "\psi_2"] & \\ E_1 \arrow[dr, "\varphi_1"] & & E_2 \arrow[dl, "\varphi_2"] \\ & G & \end{tikzcd}$$

Fig. 1 Commutative diagram for the Sieving Phase

3.3 Smoothness

Index Calculus algorithms depend on a multiplicative splitting of elements (integers, ideals or polynomials) into elements drawn from a smaller set, typically consisting of elements that are

[11] See paragraph 3.3 to understand the origin of this L notation. For the moment, just read $L_q(\alpha, c)$ as a shorthand for $\exp\left((c + o(1))(\log q)^{\alpha}(\log\log q)^{1-\alpha}\right)$.

[12] Note that those subsets are sometimes called the smoothness bases by some authors too.

in some sense considered to be *small.* Elements that do split this way are called smooth, and a fundamental problem in the analysis of Index Calculus algorithms is to estimate how the relation generation process produces those smooth elements. In most cases, the heuristic assumption is made that the elements that arise during the process essentially behave like random elements of the same size. This assumption was introduced to simplify the analysis of the algorithm and it successfully led to many algorithmic improvements and to a large number of integer factorization and discrete logarithm records.

However, depending on such a heuristic is uncomfortable and many researchers would like to come up with rigorous algorithms. Unfortunately, at the present time, the existing rigorous algorithms are much less efficient than their heuristic siblings. Quite surprisingly, the most recent advances are based on the fact that, in cases where the classical heuristic assumption become false, it is possible to use this failure to our advantage and produce more efficient algorithms.

To be more precise, let us give classic definitions and major theorems used in order to estimate these probability.

Definition 3.1. An integer is y-smooth if all its prime factors are lower than y.

Definition 3.2. A polynomial over a finite field is m-smooth if all its irreducible factors have degree lower than m.

Canfield, Erdös and Pomerance [CEP83] gave in 1983 the probability of smoothness of integers. More than a decade later, Panario, Gourdon and Flajolet [PGF98] generalized this estimation to the probability of smoothness of polynomials in finite fields. A less general result in this direction was obtained earlier in [Odl85]. These two main results that are surprisingly close can be summarized in the following estimate.

Estimate 3.1. *The probability for an arbitrary integer lower than x to be y-smooth (respectively for a random polynomial of degree less than n to be m-smooth) is:*

$$u^{-u+o(1)}$$

where $u = \dfrac{\log x}{\log y}$ *(respectively* $u = \dfrac{n}{m}$*).*

The very first analyses of the asymptotic running time of Index Calculus algorithms appeared in the 1970s and were of the form $\exp\left((c + o(1))(\log p)^{1/2}(\log\log p)^{1/2}\right)$. In fact, Index Calculus algorithms not only have in common their structure in three phases but also the expressions of their asymptotic complexities. To simplify these expressions, we usually write them with the help of the following notation:

$$L_q(\alpha, c) = \exp\left((c + o(1))(\log q)^{\alpha}(\log\log q)^{1-\alpha}\right)$$

where α and c are constants such that $0 < \alpha < 1$ and $c > 0$. This is linked to the smoothness probability of elements since it directly comes from the Estimate 3.1. The simple notation $L_q(\alpha)$ is often used when c is not specified and the expression $L_q(\alpha, c+o(1))$ is abbreviated in $L_q(\alpha, c)$ where $o(1)$ is for $q \to \infty$. The most important parameter is the first one, since it governs the transition from an exponential time algorithm to a polynomial time one. In fact, if α tends to 1, $L_q(\alpha)$ becomes exponential[13] in $\log q$ and in the other hand, if α tends to 0, $L_q(\alpha)$ becomes polynomial in $\log q$.

This notation permits not only to write the complexities in a simple an compact form but also to give an indication concerning the different ranges of application of algorithms for finite fields.

3.4 Sparse Linear Systems over Finite Fields

Index Calculus algorithms use linear algebra to recover the logarithms of the elements of the smoothness basis. Since these logarithms are determined modulo the order of the considered group,

[13] Note that, since $\log q$ is the number of bits necessary to encode elements of the group we are considering, it is the natural parameter to consider when expressing the complexity of algorithms.

we need to solve a large system of linear equations over a residue ring $\mathbb{Z}/m\mathbb{Z}$. For a long time in the 1970s and early 1980s this step was regarded as a major bottleneck, affecting the asymptotic running time estimates of algorithms. This was due to the cubic complexity of solving linear systems with classical methods such as Gaussian Elimination.

Even today, the linear algebra step remains difficult and it is a more serious problem for discrete logarithm than for factoring. The main difference is that for factoring we need solutions modulo 2, while for discrete logarithm we require solutions modulo large numbers. This is one of the reasons of the persistent gap between factorization and discrete logarithm records in $\mathbb{F}_p$, with p a prime. Fortunately, the linear systems of equations produced by Index Calculus algorithms are sparse, often to a very large extent.

A sparse matrix is a matrix that contains a relatively small number of non-zero entries. Very frequently, it takes the form of a matrix in which each line (or each column) only contains a small number of non-zero entries, compared to the dimension of the matrix. With sparse matrices, it is possible to represent in computer memory matrices with much larger dimension, describing each line (resp. column) as the list of positions containing a non-zero coefficient, together with the value of the corresponding coefficient. When dealing with a sparse linear system of equations, using plain Gaussian Elimination is a bad idea. Indeed, each pivoting step increases the number of entries in the matrix and after a relatively small number of steps, the matrix can no longer be considered as sparse. As a consequence, if the dimension of the initial matrix is large, Gaussian Elimination quickly overflows the available memory. In order to deal with sparse systems, a different approach is required.

Three main families of algorithms have been devised to deal with linear algebra in the case of sparse matrices. These methods behave better than general purpose linear algebra algorithms.

The first family, structured Gaussian Elimination, initially proposed in [Odl85] and implemented in [LO90] contains variants of the Gaussian Elimination algorithm that perform pivot selection in a way that minimizes the fill-in of the matrix throughout the algorithm. These methods are used to reduce the dimension of the original system and produce a reduced-size system which remains reasonably sparse. This reduced system is then solved using an algorithm from one of the other two families.

A common property of these two other families is that they use a matrix involved in the linear algebra in a very restrictive way. In fact, it only appears in matrix-vector products, where some variable vectors are multiplied either by the considered matrix or its transpose. The first of these two families contains Krylov Subspace Methods which have been adapted from numerical analysis and construct sequences of mutually orthogonal vectors. In particular, this family contains the Lanczos and Conjugate Gradient algorithms, already described for the discrete logarithm context in [COS86]. The second family contains the Wiedemann algorithm [Wie86] and its generalization for parallel processing, Block Wiedemann. To put it in a nutshell, the algorithms in this family find a solution of a linear system by computing the minimal polynomial[14] of the considered matrix.

Both the Krylov Subspace and Wiedemann families of algorithms cost a number of matrix-vector multiplications equal to a small multiple of the matrix dimension. Thus, for an $N \times N$ matrix containing λ entries per line on average, the global cost is $O(\lambda N^2)$.

4 Discrete Logarithm in Finite Fields

4.1 A Short History

The earliest methods introduced to compute discrete logarithms are generic. Of course, the fact that discrete logarithms can be computed using exhaustive search is self-evident. However, the algorithmic techniques to outperform this simple approach are more recent. The first method to achieve this is the Baby step/Giant step, initially introduced in 1971 by Shanks [Sha71] for the

[14] More precisely, this is the goal of Wiedemann algorithm. The block version computes something somewhat different but quite similar.

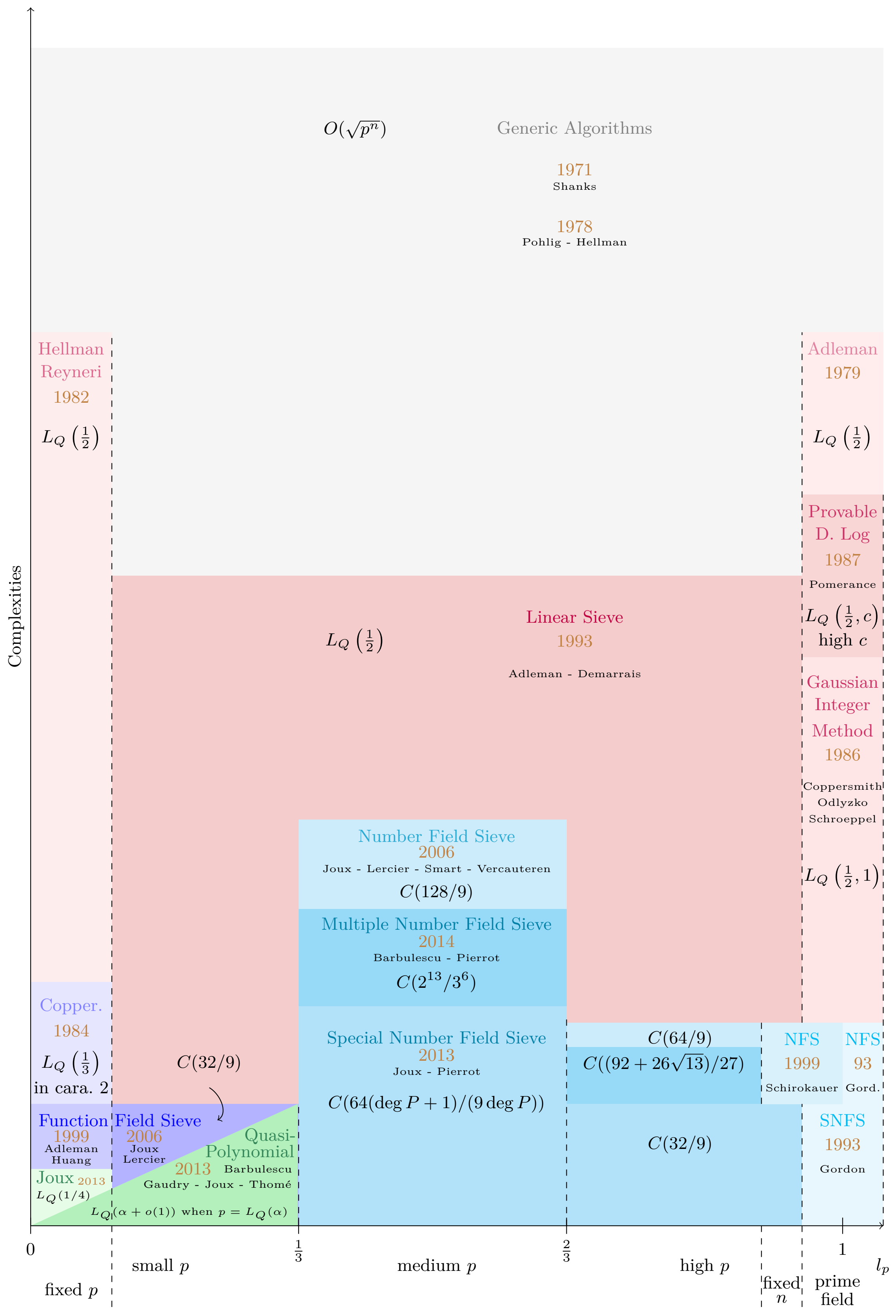

Fig. 2 Bird's-Eye View of Algorithms for Discrete Logarithm in Finite Fields and their Complexities from 1970 to 2014. We warn the lector that his drawing is not to scale since a difference in the first or in the second parameter in the L_Q notation does not have the same effect on the complexity at all. Yet, the main color of each algorithm illustrates the variation of the first parameter: we depict $L_Q(1/2)$ algorithms in red, $L_Q(1/3)$ in blue, and $L_Q(1/4)$ and quasi-polynomial in green. For a fixed color, the darker an algorithm is drawn, the more recent it is. Furthermore we introduce in this drawing the notation $C(x) = L_Q(1/3, (x)^{1/3})$.

Date	Field	Bitsize	Cost (CPU.hours)	Algorithm	Authors
1992	2^{401}	401	114000	[COS86]	Gordon, McCurley
1996	p	281	?	[COS86]	Weber, Denny, Zayer
1998/02	Special p	427	12 500	[Gor93]	Weber
1998/05	p	298	2900	[COS86]	Joux, Lercier
2001/01	p	364	290	[JL03]	Joux, Lercier
2001/04	p	397	960	[JL03]	Joux, Lercier
2001/09	2^{521}	521	2 000	[JL02]	Joux, Lercier
2002	2^{607}	607	$> 200\,000$	[Cop84]	Thomé
2005/06	p	431	350	[JL03]	Joux, Lercier
2005/09	2^{613}	613	26 000	[JL02]	Joux, Lercier
2005/10	65537^{25}	400	50	[JL06]	Joux, Lercier
2005/11	370801^{30}	556 $\star$	200	[JL06]	Joux, Lercier
2007	p	530	29 000	[JL03]	Kleinjung
2012/06	$3^{6 \cdot 97}$	923	895 000	[JL06]	Hayashi, Shimoyama, Shinohara, Takagi
2012/12	p^{47}	1175 $\star$	32 000	[Jou13a]	Joux
2013/01	p^{57}	1425 $\star$	32 000	[Jou13a]	Joux
2013/02	2^{1778}	1778 $\star$	220	[Jou13b]	Joux
2013/02	2^{1991}	1991 $\star$	2200	[GGMZ13]	Gologlu, Granger, McGuire, Zumbragel
2013/03	2^{4080}	4080 $\star$	14 100	[Jou13b]	Joux
2013/04	2^{809}	809	19 300	[AH99, JL06]	The Caramel Group
2013/04	2^{6120}	6120 $\star$	750	[GGMZ13, Jou13b]	Gologlu, Granger, McGuire, Zumbragel
2013/05	2^{6168}	6168 $\star$	550	[Jou13b]	Joux
2014/01	$3^{6 \cdot 137}$	1303	920	[Jou13b]	Adj, Menezes, Oliveira, Rodriguez-Henriquez
2014/01	2^{9234}	9234 $\star$	398 000	[Jou13b]	Granger, Kleinjung, Zumbragel
2014/01	2^{4404}	698-bit subgroup	52 000	[Jou13b]	Granger, Kleinjung, Zumbragel

Table 1 History of discrete logarithm records. The $\star$ in the bitsize column indicates (possibly twisted) Kummer extensions.

computation of class numbers in quadratic fields. The next technique, proposed in 1978, is the Pollard Rho method [Pol78], a variation on Pollard's Rho factoring algorithm [Pol75] from 1975.

Interestingly, the link that Pollard's Rho algorithm shows between factorization of integers and the computation of discrete logarithms modulo prime is much more general and most of the algorithms known to solve one of the problems admit variants that apply to the other. There are some exceptions. For example, it is not known how to obtain a variation of the Elliptic Curve factoring Method (ECM) of Lenstra to compute discrete logarithms. However, a variation of ECM can be used [MW96, JN03] to provide a relationship between the hardness of CDH and DLP. Until recently, it was believed that this relationship between the hardness of integer factorization and discrete logarithm computations could be extended to arbitrary finite fields. However, due to the recent results on the computation of discrete logarithms in small characteristic, this is no longer clear.

In 1976, the invention of Diffie–Hellman key exchange kindled renewed interest on the discrete logarithm problem in finite fields. In 1977, the discovery of RSA also renewed the interest on the integer factoring problem. At that time, the state of the art on factoring was not far in advance of what was described in a book published in 1922 by Kraitchik [Kra22], which shows how the use of quadratic forms can speed-up factorization. The same book also provides methods for the computation of discrete logarithms, however, the terminology of Kraitchik used the french work *indice* instead of discrete logarithm. This terminology spawned the name *Index Calculus* for these algorithms. The early index calculus algorithms where proposed first for prime fields $\mathbb{F}_p$ [Adl79]. They achieve complexity in $L_p(1/2, c)$ for a certain constant c. The main advantage of these initial algorithms is that they can be turned into provable version as shown in [Pom87]. Moreover, they were generalized to finite fields of the form $\mathbb{F}_{p^k}$ with fixed p by Hellman and Reyneri in [HR82].

Nonetheless, the original value of c was too high for practical application. The situation was largely improved by the Gaussian Integer Method introduced in 1986 by Coppersmith, Odlyzko and Schroeppel [COS86] which lowered the value of c to 1. At that time, it was also discovered that a variation of this algorithm obtained by replacing numbers by polynomials could be used to compute discrete logarithms in small characteristic finite fields.

A drastic change occurred in 1984 when Coppersmith proposed, in the case of characteristic 2, a heuristic algorithm with complexity $L_{2^n}(1/3)$ [Cop84]. This initial progress quickly led to the introduction of several other heuristic algorithms with $L(1/3)$ complexity both for factoring and discrete logarithms computations. A survey on the early effective implementations of these algorithms for discrete logarithms appeared in 1996 [SWD96]. For a long time, $L(1/3)$-algorithms focused on field with small characteristic, prime fields and occasionally fields of the form $\mathbb{F}_{p^k}$ for small values of k [Sch00]. The view changed in 2006, with two articles that showed that taken together, the Number Field Sieve [JLSV06] and the Function Field Sieve [AH99, JL06] are enough to cover the whole range of finite fields with heuristic $L(1/3)$ algorithms. Essentially, the result was to split the finite field in three groups, small characteristic with complexity $L(1/3, (32/9)^{1/3})$, medium characteristic with complexity $L(1/3, (128/9)^{1/3})$ and large characteristic with complexity $L(1/3, (64/9)^{1/3})$.

In 2013 and 2014, several algorithmic improvements on the complexity of discrete logarithm algorithms have appeared: two variants of the Number Field Sieve have been designed for finite fields with medium to high characteristic [JP13, BP14] and a breathtaking step forward [Jou13a, GGMZ13, Jou13b, BGJT13, GKZ14] has been made for finite fields with small characteristic. We discuss this in Section 4.2.

4.2 Current Discrete Logarithms

Current discrete logarithms algorithms for finite fields vary with the relative sizes of the characteristic and the extension degree. In order to choose the one that is well-suited for a given field $\mathbb{F}_{p^n}$, we are used to write $p = L_{p^n}(l_p, c_p)$, with $0 \leqslant l_p \leqslant 1$ and c_p a value or reasonable size (*i.e.* close to 1). As for complexity, the first parameter is the most important one in this notation. In fact, for a fixed size of finite field, when the characteristic is very small, l_p is close to 0. Conversely, when the finite field is a prime field (the extension degree is thus equals to 1) the natural choice is to set $l_p = 1$. More precisely, finite fields split in three groups:

- Finite Fields with High Characteristic, when $l_p \geqslant 2/3$.
- Finite Fields with Medium Characteristic, when $1/3 \leqslant l_p \leqslant 2/3$.
- Finite Fields with Small Characteristic, when $1/3 \leqslant l_p$.

Each case is related to one algorithm which are examined in details in the sequel. The two boundary cases when l_p equals 1/3 or 2/3 are a little bit more intricate since several algorithms are available in those cases. They will not be treated here but let us simply recall that the Function Field Sieve is still the best option for some fields in the first boundary case.

We give in Figure 3 and Figure 4 two different viewpoints of the current situation. The first figure summarizes which algorithm has to be chosen for a given finite field whereas the second one shows which sizes of field are weak compared with a given complexity. The axes of Figure 4 may seem surprising to some innocent reader. However, since $\log Q = n \log p$, it is natural to compare the size of n with the size of $\log p$ (and not with the size of p).

Medium and High Characteristic

For a finite field with medium or high characteristic, Joux, Lercier, Smart and Vercauteren presented in 2006 an adaptation of the Number Field Sieve (NFS) that has a complexity in $L_{p^n}(1/3)$. For finite fields with high characteristic, it extended the variant of Shirokauer that had the same complexity, namely $L_{p^n}(1/3, (64/9)^{1/3})$, but was available only for finite fields with fixed extension degree. The NFS as proposed in [JLSV06] is an Index Calculus algorithm that takes advantage of two representations of the finite field that rely on number fields. In a nutshell, the sieving process deals with linear polynomials and the smoothness basis consists in elements in the number fields that have norms lower than a certain predefined smoothness bound. A tricky post-process permits to associate each element of the smoothness basis to an element of the finite field.

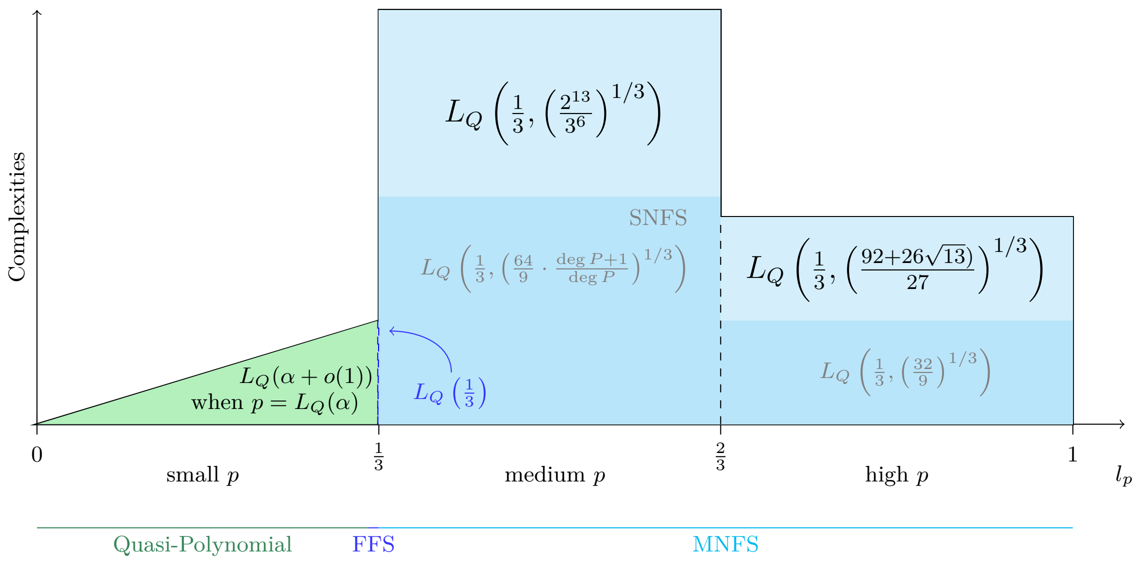

Fig. 3 Current Algorithms for Discrete Logarithms in Finite Fields and their Complexities. For a finite field of fixed size $Q = p^n$, this figure shows how asymptotic final complexities vary with the relative sizes of the characteristic p and n. We recall that we write $p = L_{p^n}(l_p)$.

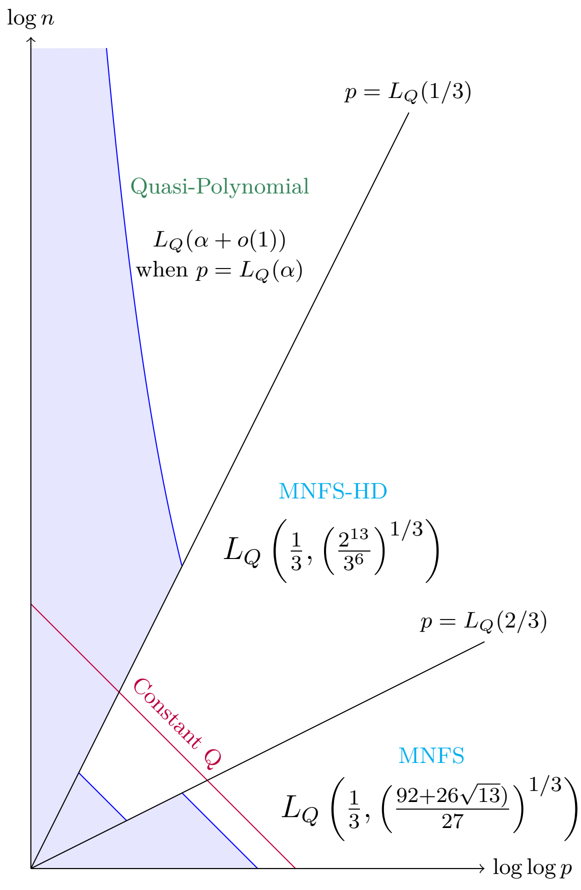
$\log n$
$p = L_Q(1/3)$
Quasi-Polynomial
$L_Q(\alpha+o(1))$
when $p = L_Q(\alpha)$
MNFS-HD
$L_Q\left(\frac{1}{3}, \left(\frac{2^{13}}{3^6}\right)^{1/3}\right)$
$p = L_Q(2/3)$
Constant Q
MNFS
$L_Q\left(\frac{1}{3}, \left(\frac{92+26\sqrt{13}}{27}\right)^{1/3}\right)$
$\log\log p$

Fig. 4 Current Domains for Discrete Logarithms Algorithms in $\mathbb{F}_{p^n}$. For each fixed size $Q = p^n$ correspond several relative sizes of p and n (see the red line) that lead to choose one algorithm or another. The blue line is an *iso-complexity line*: for a given complexity c, the DLP in each finite field that is represented in the blue part of the drawing (resp. exactly on the blue line) can be solved with an algorithm that has a complexity lower than c (resp. that equals exactly c).

For finite fields with medium characteristic, [JLSV06] proposed a variant of the classical Number Field Sieve that leads to a final complexity in $L_{p^n}(1/3, (128/9)^{1/3})$. The polynomial selection used to represent the finite field is easier, however, the sieving can no longer be done on linear polynomials. Since High Degree polynomials are used in the sieving phase, this variant of the Number Field Sieve is often called the NFS-HD. This is also the reason why the complexity of the algorithm is higher in this case than in the high characteristic case.

The currently best know algorithm for discrete logarithms in medium and high characteristic is the Multiple Number Field Sieve, a variant of NFS proposed in 2014 by Barbulescu and Pierrot [BP14]. In both cases, the main idea of MNFS is to consider not only two number fields but a lot of possible paths in the diagram. A specific benefit is obtained in the medium characteristic case since each number fields play the same role. This notion of symmetry no longer exists in the high case where one of the number field has a particular part.

Note that a Special Number Field Sieve [JP13] has been designed for both medium and high characteristic. In concerns all finite fields that have a sparse representation of their characteristic, and can be applied, so, to some finite fields coming from pairing based constructions.

Small Characteristic

For a finite field with small characteristic, namely, a field $\mathbb{F}_{p^n}$ where the characteristic can be written as $p = L_{p^n}(l, c)$ with $l \leqslant 1/3$, Joux and Lercier presented the same year an adaptation of the Function Field Sieve (FFS) that also had a complexity in $L_{p^n}(1/3)$. It was an adaptation of the FFS as introduced by Adleman in 1993. Since the beginning of 2013 a lot of things have changed for those fields with small (or extremely small) characteristics. From $L_{p^n}(1/3)$ the complexity of the DLP has dropped to $L_{p^n}(1/4 + o(1))$ [Jou13b], and finally to a heuristic quasi-polynomial algorithm [BGJT13].

Surprisingly, several of these improvements work by falsifying the standard heuristic assumptions used in older algorithms. The first of these improvements published in [Jou13a] showed that the 2006 version of the Function Field Sieve from [JL06] can be modified in a surprising way to improve its complexity. The basic idea is to slightly change how finite fields are defined and ends in a situation where the search for one smooth polynomial on the left-hand side of a relation can be amortized by constructing many possible right-hand sides from a single initial polynomial on the left. In the specific case of Kummer extensions, this can be improved further. For the first time, this new method takes advantage of the fact that the independence assumption between polynomials for the smoothness property does not hold in this context. This improvement especially focused on fields with characteristic close to $L(1/3)$ or, more generally, fields containing a subfield of size $L(1/3)$.

The next step concerns small characteristic fields, where it is possible to go well beyond the initial improvements. The basic idea can be viewed in two different ways, one can either consider a family of polynomials whose splitting probability is much higher than for random polynomials of the same degree as proposed in [GGMZ13], or start from a polynomial that splits and use a generalized version of the change of variable from [Jou13a] to construct many polynomials from this starting point. This latter approach is described in [Jou13b] and combined with a new method for computing individual logarithms, it yields a heuristic $L(1/4)$ algorithm. From an asymptotic point of view, this can be improved to a heuristic quasi-polynomial algorithm using another strategy for computing individual logarithms [BGJT13].

Note that these recent algorithms remain heuristic. However, it requires a new form of heuristic which is similar to but differs from the old one. Namely, whenever a polynomial occurs, we consider its probability of smoothness to be close to that of a random polynomial of the same degree *unless there is an explicit reason that falsifies this assumption.* Of course, whenever an explicit reason appears, by design, it largely increases the splitting probability. One of the main line of research in small characteristic is now to try to build an heuristic-free algorithm. A first step has been done in this direction in [GKZ14], removing the smoothness heuristic of the descent phase.

5 Elliptic Curve Discrete Logarithm

Subexponential Index Calculus algorithms have been developed for a variety of discrete logarithm problems. The one notable exception, where in general we still do not have algorithms better than those for the generic problem, is for elliptic curve discrete logarithms.

Most of the recent progress in discrete logarithm algorithms has come from developments in the Index Calculus method through exploitation of algebraic properties of finite fields. Unfortunately, this approach is in general not applicable to elliptic curve discrete logarithms. For elliptic curves, there exist some direct discrete logarithms algorithms that work for specific classes of curves and some indirect approaches that transfer the problem to finite fields [MOV93, FR94] or to higher genus curves [GHS02].

In general, the best known discrete logarithm algorithms for elliptic curves have exponential time complexities. However, this is not the case for higher genus curves, for which there exists Index Calculus algorithm. Moreover, some specific families of elliptic curves are also vulnerable to Index Calculus.

5.1 High genus curves

A very important result concerning curves of genus at least 3 introduced in [GTTD07] is that there exists an Index Calculus algorithm that applies to hyperellitic curves of genus $g \geq 3$ defined over $\mathbb{F}_q$ and computes discrete logarithms in time $\tilde{O}(q^{2-2/g})$. This outperforms generic algorithms whose complexity in this case is $\tilde{O}(q^{g/2})$.

Note that there are similar results concerning non-hyperelliptic curves, for example, see [EGT11, DK13].

5.2 Elliptic curves over extension fields

Where elliptic curves over extension fields are concerned, there are two main approaches: cover (or Weil descent) attacks and decomposition attacks. In addition, for some good configurations, it is possible to combine the two approaches into an even more efficient algorithm [JV12].

Weil descent

This approach introduced in [GHS02] aims at transporting the discrete logarithm problem from an elliptic curve defined over an extension field to an higher genus curve defined over a smaller field. If the genus of the target curve is not too large, this can lead to an efficient discrete logarithm algorithm.

Decomposition

The basic idea of the decomposition method [Sem04] is to find relations between the smoothness basis elements by using the n-th Semaev's summation polynomial to model the fact that n points on the curve sum to zero. Due to the symmetry of this polynomial, it is possible to reduce its degree by expressing everything in terms of the elementary symmetric polynomials in the abscissa of the solution points. Over an extension field of degree close to n above the base field, choosing the smoothness basis to be made of points with abscissa in the base field, it is possible to rewrite Semaev's polynomial as a polynomial system over the base field.

When the extension degree is much larger and cannot be decomposed into a favorable tower of extension, the situation is less clear. The typical case considered in [FPPR12, PQ12] is to take an elliptic curve over $\mathbb{F}_{2^p}$, where p is prime. The main difficulty is that, contrary to the previous setting, none of the natural choices of smoothness basis are preserved when considering the symmetric polynomials in the abscissa. As a direct consequence, it is no longer possible to easily reduce the degree of Semaev's polynomial, which makes the asymptotic behavior of the method much harder to predict.

6 The future

To date, the status of the discrete logarithm problem is quickly evolving. As a consequence, trying to predict future changes is extremely difficult. For this reason, we only sketch out the main open problems and give a small list of possible progress.

The most dangerous and notable risk for the discrete logarithm problem in general, which also applies to integer factorization, is the possibility of large scale general purpose quantum computers. If such machines were to become available, Shor algorithm would completely break these two hard problems. However, at the present time, it is unclear whether such machines will become available in the foreseeable future.

Concerning discrete logarithms in finite fields, several avenues for progress are open. First, in small characteristic, the present quasi-polynomial algorithm could be improved in many directions, with the removal of heuristic hypotheses, the improvement of the exponent in the polynomial part of the complexity and the search for a polynomial time algorithm. In larger characteristic, the methods that have been recently discovered cannot be applied directly. Moreover, these methods deeply rely on specific properties of polynomials which do not seem readily adaptable to numbers. Yet, the $L(1/3)$ complexity no longer seems to be a natural bound and one could possibly expect progress for the NFS in this range, stemming from totally new ideas. There is also a possibility for such eventual progress to improve the complexity of factoring.

The most difficult challenge for discrete logarithms is probably the search for a subexponential algorithm that would apply to general elliptic curves defined over large characteristic fields. However, even finding new Index Calculus algorithms to cover additional special cases of curves is already a very challenging and fascinating problem in this field of research.

References

[Adl79] Leonard M. Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography (abstract). In *FOCS*, pages 55–60, 1979.

[AFK89] Martín Abadi, Joan Feigenbaum, and Joe Kilian. On hiding information from an oracle. *J. Comput. Syst. Sci.*, 39(1):21–50, 1989.

[AH99] Leonard M. Adleman and Ming-Deh A. Huang. Function field sieve method for discrete logarithms over finite fields. *Inf. Comput.*, 151(1-2):5–16, 1999.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[BD94] Mike Burmester and Yvo Desmedt. A secure and efficient conference key distribution system (extended abstract). In *EUROCRYPT*, pages 275–286, 1994.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[BGJT13] Razvan Barbulescu, Pierrick Gaudry, Antoine Joux, and Emmanuel Thomé. A quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. *CoRR*, abs/1306.4244, 2013.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *J. Cryptology*, 17(4):297–319, 2004.

[BLS11] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. On the correct use of the negation map in the Pollard Rho method. In *Public Key Cryptography*, pages 128–146, 2011.

[BP14] Razvan Barbulescu and Cécile Pierrot. The multiple number field sieve for medium and high characteristic finite fields. *IACR Cryptology ePrint Archive*, 2014:147, 2014.

[CEP83] Earl Rodney Canfield, Paul Erdös, and Carl Pomerance. On a problem of Oppenheim concerning factorisatio numerorum. In *Journal of Number Theory*, volume 17, pages 1–28, 1983.

[CGH00] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *CoRR*, cs.CR/0010019, 2000.

[CHK12] Jung Hee Cheon, Jin Hong, and Minkyu Kim. Accelerating Pollard's Rho algorithm on finite fields. *J. Cryptology*, 25(2):195–242, 2012.

[Cop84] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–593, 1984.

[COS86] Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986.

[Den82] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[Den02] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In *ASIACRYPT*, pages 100–109, 2002.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DK13] Claus Diem and Sebastian Kochinke. Computing discrete logarithms with special linear systems. preprint, 2013.

[DOW92] Whitfield Diffie, Paul C. Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[EGT11] Andreas Enge, Pierrick Gaudry, and Emmanuel Thomé. An $L(1/3)$ discrete logarithm algorithm for low degree curves. *J. Cryptology*, 24(1):24–41, 2011.

[FJM13] Pierre-Alain Fouque, Antoine Joux, and Chrysanthi Mavromati. Multi-user collisions: Applications to discrete logs, Even-Mansour and prince. *IACR Cryptology ePrint Archive*, 2013:761, 2013.

[FPPR12] Jean-Charles Faugère, Ludovic Perret, Christophe Petit, and Guénaël Renault. Improving the complexity of index calculus algorithms in elliptic curves over binary fields. In *EUROCRYPT*, pages 27–44, 2012.

[FR94] Gerhard Frey and Hans-Georg Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62:865–874, 1994.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[GGMZ13] Faruk Göloglu, Robert Granger, Gary McGuire, and Jens Zumbrägel. On the function field sieve and the impact of higher splitting probabilities - application to discrete logarithms in and. In *CRYPTO (2)*, pages 109–128, 2013.

[GHS02] Pierrick Gaudry, Florian Hess, and Nigel P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *J. Cryptology*, 15(1):19–46, 2002.

[GKZ14] Robert Granger, Thorsten Kleinjung, and Jens Zumbrägel. On the powers of 2. Cryptology ePrint Archive, Report 2014/300, 2014.

[Gor93] Daniel M. Gordon. Discrete logarithms in GF(p) using the number field sieve. *SIAM J. Discrete Math.*, 6(1):124–138, 1993.

[GTTD07] Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault, and Claus Diem. A double large prime variation for small genus hyperelliptic index calculus. *Math. Comput.*, 76(257):475–492, 2007.

[HR82] Martin E. Hellman and Justin M. Reyneri. Fast computation of discrete logarithms in GF(q). In *CRYPTO*, pages 3–13, 1982.

[JL02] Antoine Joux and Reynald Lercier. The function field sieve is quite special. In *ANTS*, pages 431–445, 2002.

[JL03] Antoine Joux and Reynald Lercier. Improvements to the general number field sieve for discrete logarithms in prime fields. a comparison with the gaussian integer method. *Math. Comput.*, 72(242):953–967, 2003.

[JL06] Antoine Joux and Reynald Lercier. The function field sieve in the medium prime case. In *EUROCRYPT*, pages 254–270, 2006.

[JLSV06] Antoine Joux, Reynald Lercier, Nigel P. Smart, and Frederik Vercauteren. The number field sieve in the medium prime case. In *CRYPTO*, pages 326–344, 2006.

[JN03] Antoine Joux and Kim Nguyen. Separating decision Diffie-Hellman from computational Diffie-Hellman in cryptographic groups. *J. Cryptology*, 16(4):239–247, 2003.

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *J. Cryptology*, 17(4):263–276, 2004.

[Jou13a] Antoine Joux. Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields. In *EUROCRYPT*, pages 177–193, 2013.

[Jou13b] Antoine Joux. A new index calculus algorithm with complexity $L(1/4+o(1))$ in very small characteristic. *IACR Cryptology ePrint Archive*, 2013:95, 2013.

[JP13] Antoine Joux and Cécile Pierrot. The special number field sieve in finite fields - Application to pairing-friendly constructions. In *Pairing*, pages 45–61, 2013.

[JV12] Antoine Joux and Vanessa Vitse. Cover and decomposition index calculus on elliptic curves made practical - application to a previously unreachable curve over $\mathbb{F}_{p^6}$. In *EUROCRYPT*, pages 9–26, 2012.

[Kra22] M Kraïtchik. *Théorie des nombres*. Gauthier–Villars, 1922.

[KS01] Fabian Kuhn and René Struik. Random walks revisited: Extensions of Pollard's Rho algorithm for computing multiple discrete logarithms. In *Selected Areas in Cryptography*, pages 212–229, 2001.

[LO90] Brian A. LaMacchia and Andrew M. Odlyzko. Solving large sparse linear systems over finite fields. In *CRYPTO*, pages 109–133, 1990.

[MOV93] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[MW96] Ueli M. Maurer and Stefan Wolf. Diffie-Hellman oracles. In *CRYPTO*, pages 268–282, 1996.

[Odl85] Andrew M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. *Advances in Cryptology: Proceedings of EUROCRYPT 84*, 209:224–314, 1985.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[PGF98] Daniel Panario, Xavier Gourdon, and Philippe Flajolet. An analytic approach to smooth polynomials over finite fields. In *ANTS*, pages 226–236, 1998.

[PH78] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Pol75] John Pollard. A Monte Carlo method for factorization. In *BIT Numerical Mathematics*, pages 331–334, 1975.

[Pol78] John Pollard. Monte Carlo methods for index computations mod p. In *Mathematics of Computation*, pages 918–924, 1978.

[Pom87] Carl Pomerance. Fast, rigorous factorization and discrete logarithm algorithms. In *Discrete algorithms and complexity*, pages 119–143, 1987.

[PQ12] Christophe Petit and Jean-Jacques Quisquater. On polynomial systems arising from a Weil descent. In *ASIACRYPT*, pages 451–466, 2012.

[QD89] Jean-Jacques Quisquater and Jean-Paul Delescaille. How easy is collision search. new results and applications to DES. In *CRYPTO*, pages 408–413, 1989.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[Sch00] Oliver Schirokauer. Using number fields to compute logarithms in finite fields. *Math. Comput.*, 69(231):1267–1283, 2000.

[Sem04] Igor Semaev. Summation polynomials and the discrete logarithm problem on elliptic curves. *IACR Cryptology ePrint Archive*, 2004:31, 2004.

[Sha71] Daniel Shanks. Class number, a theory of factorization and genera. In *Proc. Symp. Pure. Math*, pages 415–440, 1971.

[Sho97a] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[Sho97b] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[SWD96] Oliver Schirokauer, Damian Weber, and Thomas F. Denny. Discrete logarithms: The effectiveness of the index calculus method. In *ANTS*, pages 337–361, 1996.

[Tes00] Edlyn Teske. On random walks for Pollard's Rho method. *Mathematics of Computation*, 70:809–825, 2000.

[vOW99] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *J. Cryptology*, 12(1):1–28, 1999.

[Wie86] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, 1986.